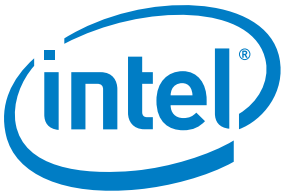


Closed Loop Platform Automation – Workload Resiliency

User Guide

April 2019

Document Number: 338936-001

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at http://www.intel.com/ or from the OEM or retailer.

No computer system can be absolutely secure.

Intel, the Intel logo, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.

* Other names and brands may be claimed as the property of others.

Contents

Figures

Tables

Revision History

Date	Revision	Description
April 2019	001	Initial release.

1.0 Introduction

With the massive influx of devices coming with 5G and IOT, there will be tremendous pressure on next generation infrastructure. The network must scale to support billions of devices, and a wide variety of use cases. Meeting these demands requires response times in many areas of the network that exceed the capabilities of manual processes. The industry is recognizing it is time to accelerate the automation journey.

The relationship between automated operations and excellent customer experience is an area of significant interest to Service Providers. The closed loop platform resiliency demo described in this document leverages the integration around orchestration, monitoring software, and platform resource provisioning with the help of *Closed Loop Platform Automation* that:

- Improves customer experience.
- Automates of operational processes and tasks, along with root cause analysis.
- Manages complexity and enables operations to meet upcoming efficiency demands.
- Reduces capital expenses and operating expenses.
- Identifies service level agreement violations.

This document provides instructions to create a closed loop resiliency demo that minimizes network outage time and therefore maximizes service availability. It shows that by using Intel® architecture platform specific metrics and events, we can monitor the health of the platform and identify issues that may impact the end-user experience. We use these key indicators to raise alarms and trigger the correct remediation that prevents downtime for the customer.

Note: This document uses a virtual Broadband Network Gateway (vBNG) reference application to showcase the resiliency of the workload with closed loop failover capability, however the solution itself is application-agnostic.

This document is part of the Network Transformation Experience Kit, which is available at: https://networkbuilders.intel.com/

1.1 Terminology

Table 1. Terminology

Term	Description
Aodh	OpenStack* alarming service project (not an acronym)
DPDK	Data Plane Development Kit
MCE	Machine Check Exception
RAS	Reliability Availability Serviceability
SNMP	Simple Network Management Protocol
TSDB	Time Series Database
vBNG	Virtual Broadband Network Gateway
VES	VNF Event Stream

1.2 Reference documents

Table 2. Reference documents

Ref	Document	Location
[1]	Collectd Metrics and Events	https://wiki.opnfv.org/display/fastpath/Collectd+Metrics+and+Events
[2]	Intel® Run Sure Technology	https://www.intel.com/content/www/us/en/architecture-and-technology/intel-run-sure-technology.html
[3]	Linux* machine check exception injection tool: MCE-Inject	https://github.com/andikleen/mce-inject/
[4]	Barometer repository (containing chosen scripts from guide)	https://gerrit.opnfv.org/gerrit/gitweb?p=barometer.git

2.0 System Overview

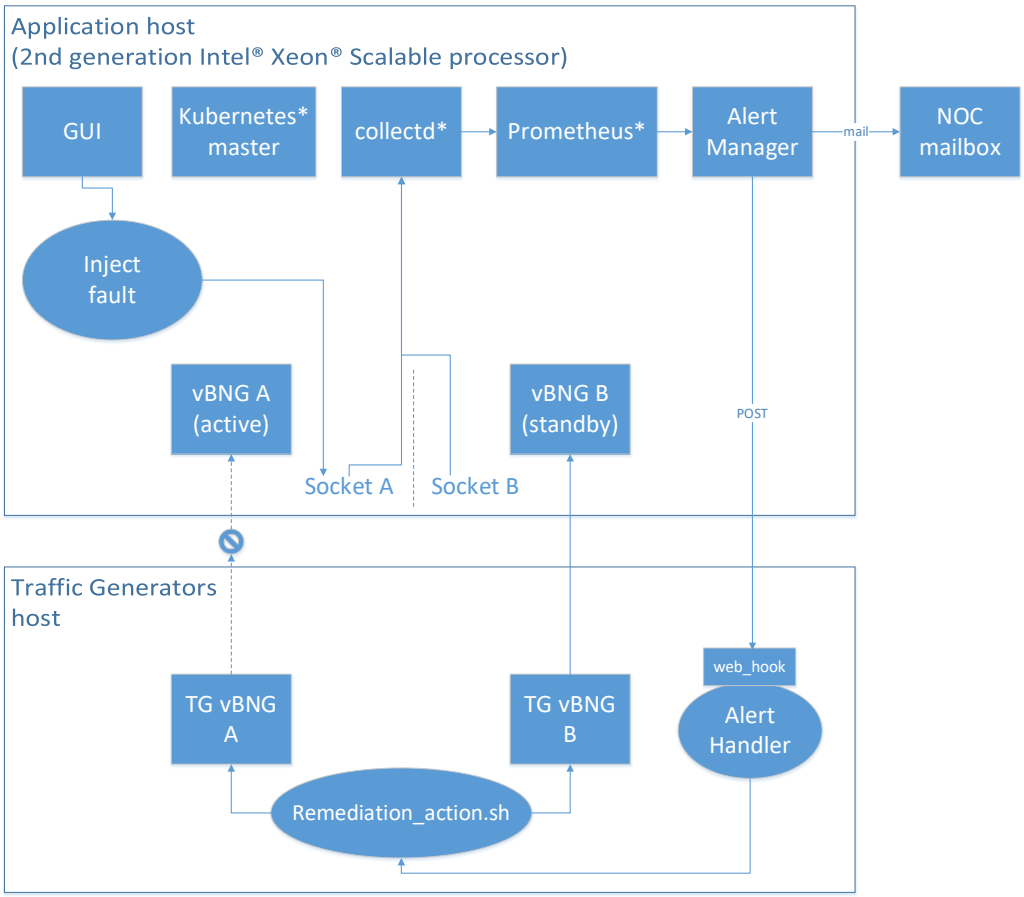
This section describes the architecture of the solution and the requirements for hardware, software, and BIOS.

Note: This document uses a virtual Broadband Network Gateway (vBNG) reference application to showcase the resiliency of the workload with closed loop failover capability, however the solution itself is application-agnostic.

2.1 Setup architecture

The (vBNG) application is initially deployed in a "warm standby" model. We trigger a platform fault that is reflected via a platform metric. As a part of the closed loop error detection and correction, the traffic will be switched from the active to the standby application to maintain an uninterrupted service.

Figure 1. Closed loop resiliency setup



We run two application instances at the start of the demonstration [A] active and [B] standby, on a single server platform, each on a separate socket. Both traffic generators are configured and providing traffic to both sockets. The application instances, created by Kubernetes, will start processing traffic immediately the application instance becomes active.

Platform telemetry, such as resource utilization, status, and faults, is provided by Collectd*. Collectd is an open source collection daemon which publishes metrics to consumers using a plugin architecture. Collectd publishes metrics to many standard interfaces including VES, SNMP, SYSLOG/Logstash, Ceilometer, Gnocchi, Aodh, Kafka/influxdb and Prometheus. A full list of metrics and events is listed in Ref [1] in Table 2. The OPNFV* Barometer project leverages Collectd and provides improvements and references to Collectd integration with Time Series Database (TSDB) and visualization tools. Telemetry for the demonstration platform is published using Prometheus.

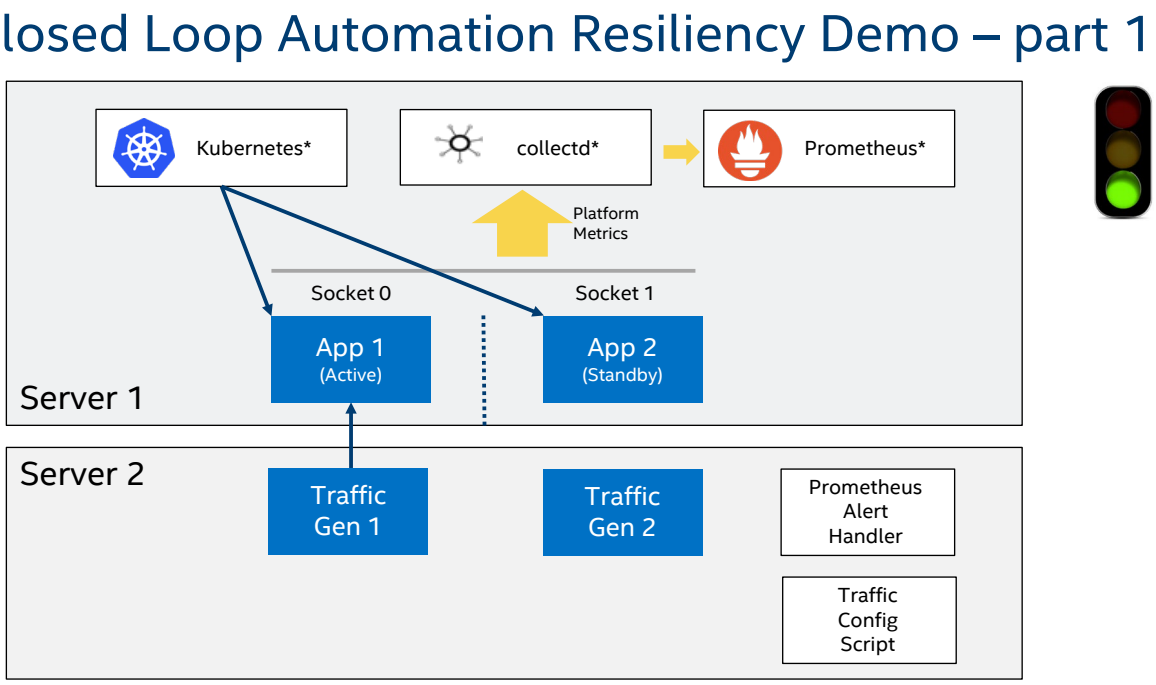
The demonstration is comprised of 3 parts.

Part 1: There are two servers. Server 1 hosts two application instances deployed by Kubernetes. Application instance 1 is in active mode receiving traffic on socket 0, the other application instance 2 is in standby on socket 1. Collectd is also running on this server as the metric collection daemon of choice. This is integrated with Prometheus, which is a monitoring and time series database solution. Prometheus has a component called alert manager which handles alerts sent by client applications, such as the Prometheus server in this case.

Server 2 hosts the traffic generator for the two application instances. It also hosts the custom alert handler. The alert handler implements automated responses to telemetry-based alerts it receives from Prometheus.

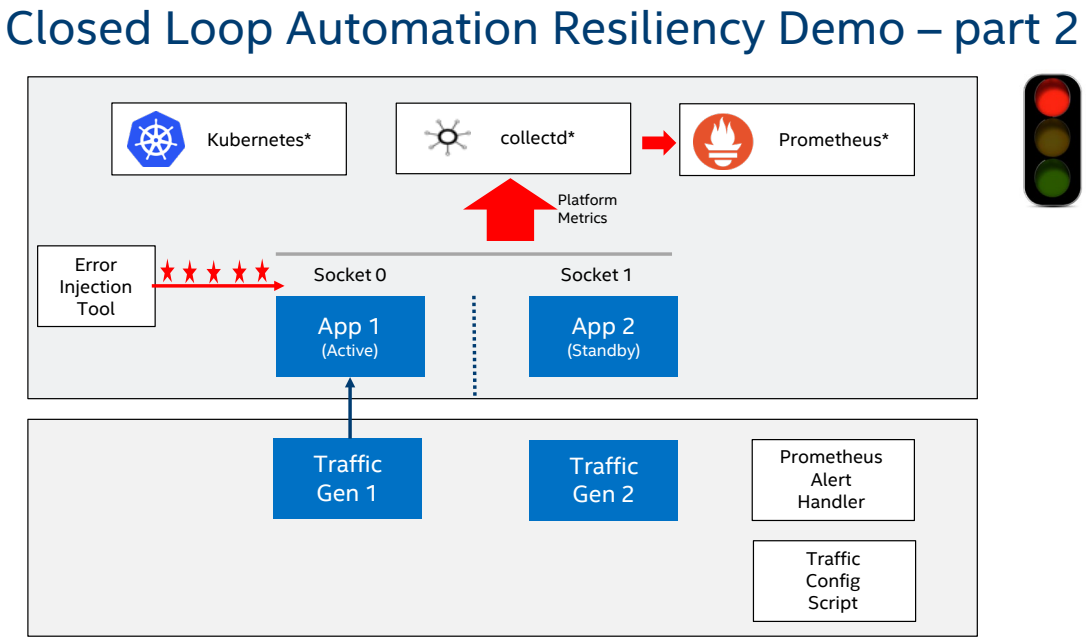
Figure 2 shows that the current state is traffic is running and there are no errors to report on the setup.

Figure 2. Default state



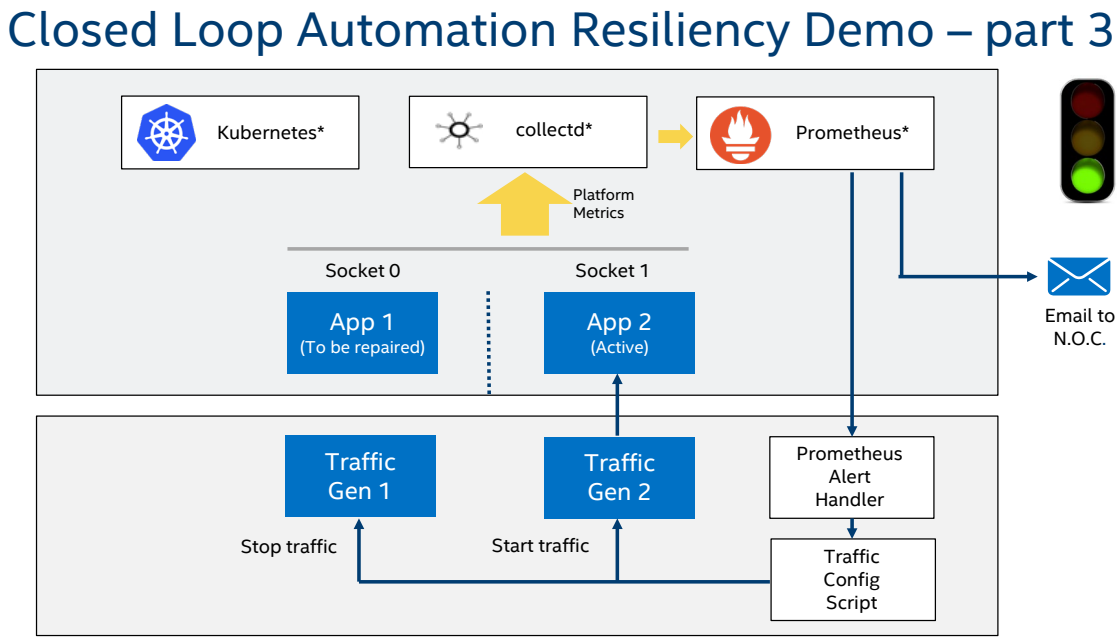
Part 2: We trigger the error scenario by injecting an Intel® architecture specific Reliability Availability Serviceability (RAS) fault [Ref 2 in Table 2], which will incur a DIMM failure on socket 0. Using an MCE error injection tool [Ref 3 in Table 2], we inject 5 correctable memory errors. This counter is picked up by Collectd and sent to Prometheus where the alert manager identifies the failure scenario. We have set a threshold in the Prometheus alert manager that if we receive at least 5 errors in the last 5 seconds, an alarm is raised to alert the NOC via email that a DIMM requires maintenance. The alarm also triggers the remediation action. Refer to Figure 3.

Figure 3. Error state



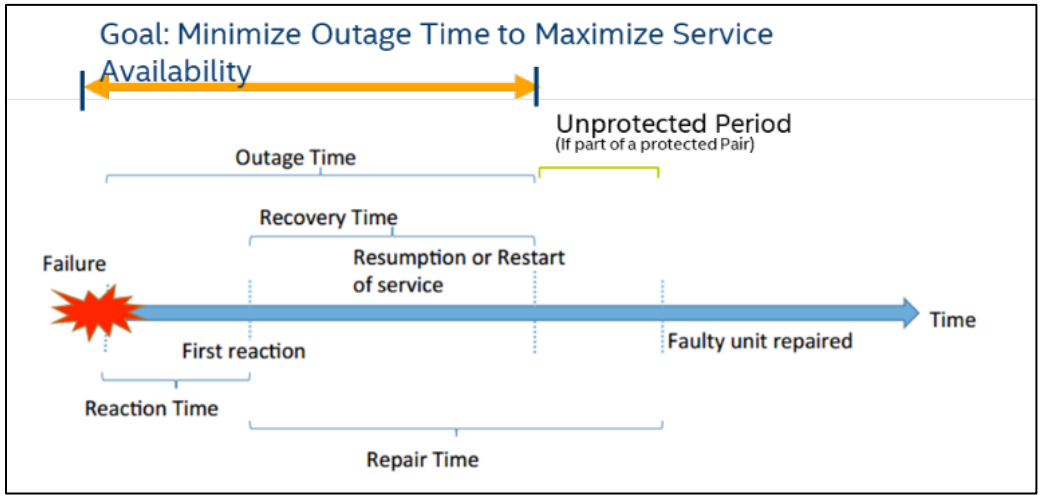
Part 3: Prometheus sends a notification to Prometheus alert handler to indicate the threshold has been breached. On receiving a JSON packet from Prometheus describing the alert, PAH calls a traffic config script, quickly adapting the system to the change in its state. In this instance, the remediation action is to stop traffic on application instance 1 and start it on application instance 2, service resumes as normal, and there is minimal disruption to the service for the customer. Refer to Figure 4.

Figure 4. Corrected failover state



The key system metrics for Resiliency and Availability scenarios are indicated in Figure 5. The goal of efficient resiliency closed loops is to minimize the outage time, which in turn maximizes the availability of the service. System Metrics include Reaction Time (includes detection), Repair Time, Recovery Time, and Outage Time. The unprotected period refers to period in which there is no 'dual active/standby' because the faulty equipment has not been replaced. The demo provides a visual of the recovery time. While still a short period, the recovery time has not been optimized for speed, which is beyond the scope of this document.

Figure 5. Recovery time calculation



2.2 Hardware requirements

Table 3. Hardware requirements

Hardware component	Server 1	Server 2
CPU	Intel® Xeon® Platinum 8276 CPU @ 2.20GHz	Intel® Xeon® Platinum 8276 CPU @ 2.20GHz
CPU(s)	112	112
Thread(s) per Core	2	2
Core(s) per Socket	28	28
Socket(s)	2	2
Memory	196798 MB	181249 MB
NIC	2 x Intel® Ethernet Network Adapter XXV710-DA2	2 x Intel® Ethernet Network Adapter XXV710-DA2
NIC Mbps	25000	25000

2.3 Software requirements

Table 4. Software requirements

Software component	Details
Operating system (OS)	Ubuntu* 18.04.1 LTS
Kernel Version	4.15.0-43-generic
Software Version	DPDK-18.11
Application	We used a virtual Broadband Network Gateway (vBNG) reference application, however the solution itself is application-agnostic.
Docker*	18.06.1-ce
Kubernetes*	V1.11.0
I40e	2.7.12
I40evf	3.6.10

2.4 BIOS configuration

Table 5. BIOS configuration

BIOS parameter	Optimized configuration
Hyper Threading (Advanced>Processor configuration)	<Enabled>
Intel® VT Directed I/O (Advanced>Integrated I/O Configuration)	<Enabled>
CPU Power and Performance (Advanced > Power n Performance)	Performance
Intel Turbo boost Technology (Advanced>Power n Performance>CPU P State Control)	Disabled
Enhanced Intel Speed step Technology (Advanced>Power n Performance>CPU P State Control)	Enable
Package C-State (Advanced>Power n Performance>CPU C State Control)	C0/C1 State

3.0 System Setup

The demo is application-agnostic. For this demo, we used a DPDK based vBNG reference application that leverages a DPDK IP pipeline application.

3.1 Application setup

The reference application was set up in the following manner:

Core allocation:

- root@<Hostname>:~# lscpu | grep NUMA
- NUMA node(s): 2
- NUMA node0 CPU(s): 0-27,56-83
- NUMA node1 CPU(s): 28-55,84-111
- Downlink Cores: ("3,59" "4,60" "31,87" "32,88")
- Uplink Cores: ("5" "61" "6" "62" "33" "89" "34" "90")

  Note: Uplink Instances are stacked two per core, 1 on the physical core and the second one on the hyper-threaded pair. Example from above ("5" "61") shows one UL on core 5 and one UL on its hyper-threaded pair 61.

- Core assignments are made in the application configuration file. Allocation for the application instances in this scenario are listed below. Note that these allocations may vary, because the core allocation is managed by the CPU Manager for Kubernetes, which takes the cores from available pool of cores.

  Instance_0_0 (Instance A: Socket 0): DL: 3, 59 UL: 5, 61 Instance_1_0 (Instance B: Socket 1): DL: 31, 87 UL: 33,89

SR-IOV port allocation:

- PCIe addresses:

  Host-config.sh is preconfigured for a system with 8PFs. Add PCIe addresses of NIC PF and VFs for the local environment.

  Note: For NIC PFs, only specify the ports that are physically present.

```
# NIC PF ports to use (2 x NIC devices, 4 x PF's)
# NIC VF ports to use for Control Plane, Downlink & Uplink
Instances (3 VFs per PF)
1 VF for Control Plane per PF
1 VF for Downlink per PF
1 VF for Uplink per PF

export NB_VFS=3
```

- Application Instance Ports:

  Add the ports used for running each Application Instance. These are used to run and attach to each instance.

```
# downlink ports
declare -a pipeline_ports_dl=("8086" "8087" "8088" "8089" )
# uplink ports
declare -a pipeline_ports_ul=("8094" "8095" "8096" "8097" )
```

  Configure Cores to be used for running Application Instances:

```
# Master Core
        declare -a master_core=0
        NOTE: Master Core is CPU (0/1) and SKU specific
```

3.2 Kubernetes* installation

On Server 1, perform the following steps as root user.

1. Application Server Preparation & Component Installation

```
# If you installed the OS via Lab Express provisioning, then your
# machine hostname and FQDN is most likely configured
incorrectly.
```

2. Set hostname and configure the host name resolution.

```
hostnamectl set-hostname <Your server1 hostname>
cat >/etc/hosts <<EOL
127.0.0.1       localhost.localdomain localhost
127.0.1.1       <Your server1 hostname>

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
EOL

# Confirm that the hostname and FQDN have been set
hostname
hostname -f
```

3. Confirm Set Proxies, when your system is behind a network proxy.

```
export http_proxy="ADD YOUR PROXY HERE"
export https_proxy=$http_proxy
export my_ip=$(ip addr show | grep "First 2 octets of IP address,
eg: 192.168" | awk '{print $2}')
export no_proxy=$(hostname),$(hostname -f),$my_ip,127.0.0.1
```

4. Download packages required to build CNI plugins.

```
cd  $MYHOME/k8s/

# Download the required package. We used Go from:
https://dl.google.com/go/go1.9.7.linux-amd64.tar.gz

tar xzvf go1.9.7.linux-amd64.tar.gz
export PATH=$PATH:`pwd`/go/bin
```

5. Install Kubernetes.

   Note: If your server is behind a network proxy, be sure to configure the proxy accordingly.

   a. Install Docker.

```
# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
# sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu bionic stable"
# sudo apt -y install docker-ce=18.06.1~ce~3-0~ubuntu
```

   b. Install Kubernetes.
   Use the installation procedure that is specific to your platform.

   c. Verify that the services below are running correctly.

   etcd.service - Etcd Server

```
Loaded: loaded (/lib/systemd/system/etcd.service; enabled; vendor
preset: enabled)
Active: active (running) since Tue 2018-11-06 12:49:01 GMT; 21h
ago
```

   kube-apiserver.service - Kubernetes API Server

```
Loaded: loaded (/lib/systemd/system/kube-apiserver.service;
enabled; vendor preset: enabled)
Active: active (running) since Tue 2018-11-06 12:49:10 GMT; 21h
ago
```

   kube-scheduler.service - Kubernetes Scheduler Plugin

```
Loaded: loaded (/lib/systemd/system/kube-scheduler.service;
enabled; vendor preset: enabled)
Active: active (running) since Tue 2018-11-06 16:05:25 GMT; 18h
ago
```

   kube-controller-manager.service - Kubernetes Controller Manager

```
Loaded: loaded (/lib/systemd/system/kube-controller-
manager.service; enabled; vendor preset: enabled)
Active: active (running) since Tue 2018-11-06 16:05:51 GMT; 18h
ago
```

   docker.service - Docker

```
Loaded: loaded (/lib/systemd/system/docker.service; enabled;
vendor preset: enabled)
Drop-In: /etc/systemd/system/docker.service.d
          └─http-proxy.conf
Active: active (running) since Tue 2018-11-06 12:49:27 GMT; 21h
ago
```

kubelet.service - Kubernetes Kubelet Server

```
Loaded: loaded (/lib/systemd/system/kubelet.service; enabled;
vendor preset: enabled)
Active: active (running) since Tue 2018-11-06 12:49:27 GMT; 21h
ago
```

flanneld.service - Flanneld overlay address etcd agent

```
Loaded: loaded (/lib/systemd/system/flanneld.service; enabled;
vendor preset: enabled)
Active: active (running) since Tue 2018-11-06 12:49:23 GMT; 21h
ago
```

kube-proxy.service - Kubernetes Kube-Proxy Server

```
Loaded: loaded (/lib/systemd/system/kube-proxy.service; enabled;
vendor preset: enabled)
Active: active (running) since Tue 2018-11-06 12:49:11 GMT; 21h
ago
```

d. Change directory to `cd $MYHOME/k8s`.

3.3 Install CPU Manager for Kubernetes*

In our setup, we used CPU Manager for Kubernetes* version 1.3.0.

```
# We assume that isolated CPU's and hugepages kernel options have
been
# set in the GRUB_CMDLINE_LINUX_DEFAULT line. GRUB line example:
# Pods tailored to use 2K Huge Pages - 2GB Per Pod required
GRUB_CMDLINE_LINUX_DEFAULT="default_hugepagesz=2M hugepagesz=2M
hugepages=20024 ipv6.disable=1 intel_pstate=disable rhgb
intel_iommu=on iommu=pt isolcpus=3-27,59-83,31-55,87-111
nr_cpus=112 intel_pstate=disable"
```

a. Install CPU Manager for Kubernetes (core pinning and isolation).

```
git clone  https://github.com/intel/CPU-Manager-for-
Kubernetes.git
git checkout 4607e629d3034266e129250c1d7edd341a9f8a6bq
cd CPU-Manager-for-Kubernetes
```

b. Add proxies, for example:

```
vim Dockerfile
-----------------------------------------------
FROM python:3.4.6

## ADD HTTP PROXY HERE ##
#ENV http_proxy=
## ADD HTTPS PROXY HERE ##
#ENV https_proxy=

ADD requirements.txt /requirements.txt
```

```
RUN pip install -r /requirements.txt

ADD . /cmk
WORKDIR /cmk

RUN chmod +x /cmk/cmk.py

RUN tox -e lint
RUN tox -e unit
RUN tox -e integration
RUN tox -e coverage

RUN /cmk/cmk.py --help && echo ""

ENTRYPOINT [ "/cmk/cmk.py" ]
-----------------------------------------------
```

c. Build Docker image.

```
make
cd ..
```

d. Install RBAC Rules and Service Account.

1. Remove any previous built cmk pods.

```
./scripts/remove-cmk.sh hostname
```

2. Create RBAC rules.

```
kubectl delete -f CPU-Manager-for-
Kubernetes/resources/authorization/cmk-rbac-rules.yaml
kubectl create -f CPU-Manager-for-
Kubernetes/resources/authorization/cmk-rbac-rules.yaml
```

3. Create Service Account.

```
kubectl delete -f CPU-Manager-for-
Kubernetes/resources/authorization/cmk-serviceaccount.yaml
kubectl create -f CPU-Manager-for-
Kubernetes/resources/authorization/cmk-serviceaccount.yaml
# NOTE:
 - If the following Error occurs:
"The connection to the server IP:6443 was refused - did you
specify the right host or port?"
 - Solve by removing the old config:     rm -rf ~/.kube/config
```

e. Edit CPU-Manager-for-Kubernetes/resources/pods/cmk-cluster-init-pod.yaml.

1. Find the following line:

```
"/cmk/cmk.py cluster-init --host-list=node1,node2,node3 --
saname=cmk-serviceaccount --cmk-img-pol=IfNotPresent"
```

2. Replace `--host-list=node1,node2,node3` with `--all-hosts`

3. The following should correspond to the isolcpu settings:

```
--num-exclusive-cores=<num>  Number of data plane cores [default:
4].
--num-shared-cores=<num>  Number of control plane cores [default:
1].
```

4. Add `--exclusive-mode=spread --shared-mode=spread`. If you have spread enabled, it will allow you to equally assign cores across NUMA nodes.

```
 # E.g.:
root@Server1:~/vBNG/k8s# cat CPU-Manager-for-
Kubernetes/resources/pods/cmk-cluster-init-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: cmk-cluster-init-pod
  name: cmk-cluster-init-pod
  #namespace: user-supplied-namespace
spec:
  serviceAccountName: cmk-serviceaccount
  containers:
  - args:
      # Change this value to pass different options to cluster-
init.
      - "/cmk/cmk.py cluster-init --exclusive-mode=spread --
shared-mode=spread --all-hosts --num-exclusive-cores=20 --num-
shared-cores=2 --saname=cmk-serviceaccount --cmk-img-
pol=IfNotPresent"
    command:
    - "/bin/bash"
    - "-c"
    image: cmk:v1.3.0
    name: cmk-cluster-init-pod
  restartPolicy: Never
root@<Hostname>:~/vBNG/k8s#
root@Server1:~/vBNG/k8s#
```

f. Modify the API Server.

```
cd CPU-Manager-for-Kubernetes
vim /etc/kubernetes/apiserver
```

1. Ensure it contains MutatingAdmissionWebhook,ValidatingAdmissionWebhook as shown in the example below:

```
KUBE_ADMISSION_CONTROL="--admission-
control=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStor
ageClass,ResourceQuota,DefaultTolerationSeconds,MutatingAdmission
Webhook,ValidatingAdmissionWebhook"
```

2. Restart services.

```
systemctl restart kube-apiserver kube-scheduler kube-controller-
manager kube-proxy kubelet
```

g. Remove Taints.

```
cd ..
```

1. Retrieve node name.

```
kubectl get nodes
```

```
------------------------------------------------------------------
---
NAME                            STATUS    ROLES     AGE
VERSION
<hostname>   Ready     <none>    1d        v1.11.0
------------------------------------------------------------------
---
kubectl taint nodes  <hostname> cmk-
kubectl create  -f CPU-Manager-for-Kubernetes/resources/pods/cmk-
cluster-init-pod.yaml
# NOTE: If issues occur with cmk-cluster init Pods not being
# able to connect to the api-server run following commands:
# (flush iptables)
                systemctl stop docker
                systemctl stop flanneld
                systemctl stop kube-proxy
                systemctl stop kubelet
                iptables -P INPUT ACCEPT
                iptables -P FORWARD ACCEPT
                iptables -P OUTPUT ACCEPT
                iptables -F
                iptables -X
                iptables -t nat -F
                iptables -t nat -X
                iptables -t mangle -F
                iptables -t mangle -X
                iptables -t raw -F
                iptables -t raw -X
                systemctl restart docker
                systemctl restart flanneld
                systemctl restart kube-proxy
                systemctl restart kubelet
```

h. Check if cores are isolated.

```
kubectl logs pod/cmk-init-install-discover-pod-<hostname> init
```

i. Check if all CMK components are deployed.

```
kubectl get all --all-namespaces
------------------------------------------------------------------
------------------------------------------------
NAMESPACE   NAME
READY     STATUS      RESTARTS   AGE
default     pod/cmk-cluster-init-pod
0/1       Completed   0          1m
default     pod/cmk-init-install-discover-pod-<hostname>
0/2       Completed   0          1m
default     pod/cmk-reconcile-nodereport-ds-<hostname>-mzd8m
2/2       Running     0          1m
default     pod/cmk-webhook-podod
1/1       Running     0          50s
------------------------------------------------------------------
------------------------------------------------
```

6. Install Device Plugin.

```
# Assuming PF ports have been bound to DPDK's igb_uio driver and
3 VF's
# per PF have been created and bound to igb_uio:
```

a. Create `mkdir /etc/pcidp`.

1. Populate Downlink-rootDevice below to match host-config.sh vf_ports_dl.
2. Populate Uplink-rootDevice below to match host-config.sh vf_ports_ul.

```
cat > /etc/pcidp/config.json <<EOL
root@Server1:~/vBNG# cat /etc/pcidp/config.json
{
    "resourceList":
    [
        {
            "resourceName": "Downlink_s0",
            "rootDevices": ["0000:18:02.1"],
            "sriovMode": false,
            "deviceType": "uio"
        },
        {
            "resourceName": "Uplink_s0",
            "rootDevices": ["0000:18:02.2"],
            "sriovMode": false,
            "deviceType": "uio"
        },
        {
            "resourceName": "Downlink_s1",
            "rootDevices": ["0000:af:02.1"],
            "sriovMode": false,
            "deviceType": "uio"
        },
        {
            "resourceName": "Uplink_s1",
            "rootDevices": ["0000:af:02.2"],
            "sriovMode": false,
            "deviceType": "uio"
        }
    ]
}
root@Server1:~/vBNG#

    EOL
```

b. Build and run `cd docker-image-pci-dev-plugin`.

1. Before running docker build, ensure your proxies are still set.
2. Run the following command (it may take a while to build image):

```
docker build -t pci-device-plugin-image . --build-arg
http proxy=$http proxy --build-arg https proxy=$https proxy
# NOTE:
 - If following error occurs:
```

```
   "Get https://registry-1.docker.io/v2/: net/http: request
canceled while waiting for connection (Client.Timeout exceeded
while awaiting headers)"
 - Add your proxies to /etc/systemd/system/docker.service.d/http-
proxy.conf
```

c. Check if image created succesfully.

```
docker images
--------------------------------------------------------------------
-------------------------
REPOSITORY                TAG                 IMAGE ID
CREATED             SIZE
pci-device-plugin-image   latest              aebdf60c4a1f
29 minutes ago      924MB
--------------------------------------------------------------------
-------------------------
```

d. Create Daemonset.

```
kubectl create -f pcidp-daemonset-pktgen.yaml
cd ..
```

e. Check that the device plugin is running.

```
kubectl get all --all-namespaces
--------------------------------------------------------------------
-----------------------------------------------------
NAMESPACE   NAME
READY     STATUS      RESTARTS   AGE
default     pod/cmk-cluster-init-pod
0/1       Completed   0          1h
default     pod/cmk-init-install-discover-pod-<hostname>
0/2       Completed   0          1h
default     pod/cmk-reconcile-nodereport-ds-<hostname>-mzd8m
2/2       Running     0          1h
default     pod/cmk-webhook-pod
1/1       Running     0          1h
default     pod/pci-device-plugin-xt9hg
1/1       Running     0          21s

NAMESPACE   NAME                            TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)   AGE
default     service/cmk-webhook-service   ClusterIP
10.254.242.123   <none>        443/TCP   1h
default     service/kubernetes            ClusterIP   10.254.0.1
<none>        443/TCP   1d

NAMESPACE   NAME
DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
default     daemonset.apps/cmk-reconcile-nodereport-ds-<hostname>
1         1         1         1            1           <none>
1h
default     daemonset.apps/pci-device-plugin
--------------------------------------------------------------------
--------------------------------------------------------------------
------------------------------
```

```
# NOTE: Both CMK & pci-device-plugin should now be fully
operational
```

7. Create NFS.

    a. If you do not have nfs-kernel-server installed:

```
sudo apt-get install nfs-kernel-server
mkdir /test
chmod 777 /test
```

    b. Create Kubernetes master files and copy to NFS folder.

```
cd ..
cd cli_generation/
vim config-generate.ini
```

    c. Modify inst_path to match your $MYHOME directory.

    d. Generate config files for your application Pods.

    e. Copy configs to NFS folder.

```
cd $MYHOME/<Your application>-config/kubernetes/
cp -r dl_master/ ul_master/ /test
cd ..
cp -r <Your application>_configs/ /test/
```

    f. Add following lines to /etc/exports

```
/test *(rw,sync,no_subtree_check)
exportfs -a
```

    g. Make sure NFS is running.

```
service nfs-kernel-server status
```

    h. Modify nfs/nfs_volume.yaml to include your host name.

```
cd $MYHOME/k8s
vim nfs/nfs_volume.yaml
# E.g.:
nfs:
server: <hostname>
path: "/test
```

    i. Create nfs volume.

```
kubectl create -f nfs/nfs_volume.yaml
```

3.4 Build images

1. Build all images.
2. Ensure images are built.

```
docker images
------------------------------------------------------------------
--------------------------------
REPOSITORY                          TAG                 IMAGE ID
CREATED             SIZE
vbng-cloud-pktgen-init          latest              8957d8f4089e
57 seconds ago      259MB
vbng-cloud-vbng-init            latest              8957d8f4089e
57 seconds ago      259MB
```

```
vbng-pktgen                     latest              8957d8f4089e
57 seconds ago      259MB
vbng-pod                        latest              8957d8f4089e
57 seconds ago      259MB
-------------------------------------------------------------------
------------------------------
```

3.5 Modify and run workload

Note: This example uses a virtual Broadband Network Gateway (vBNG) reference application to showcase the resiliency of the workload with closed loop failover capability, however the solution itself is application-agnostic.

1. Set up PF-Init for the application.
   a. Change to socket_0 or socket_1 directory depending on which NUMA node you want to run on your Applications on.

```
# Socket 0 Example:
# The first 3 NUMA node cores are used as follows:
# - core 0:system, core 1:cli-core, core 2:pf-init-core
```

2. Label Node to Application.

```
kubectl get node (get your Node name)
-------------------------------------------------------------------
---
NAME                            STATUS    ROLES     AGE
VERSION
<hostname>   Ready     <none>    2h        v1.11.0
-------------------------------------------------------------------
---
kubectl label node <hostname> application=true
```

3. Set up application pods.

```
# Socket 0 Example:
```

   a. Ensure the following is set as described below:

```
vim vbng-pods/socket_0/k8s-application-pod-config-0-X-ul.yaml &&
vim k8s-application-pod-config-0-X-dl.yaml
```

4. Set up routes.

   Create specific routes for ingress/egress traffic on your application pods.

5. Run pf_init and application instances.

   Pass socket number to script to launch instances on that socket based on the application you run on each socket.

3.6 Collectd

1. Apply Collectd patch.

```
Note: dpdk_telemetry patch is not yet open sourced. It will be
available soon.
#git clone https://github.com/collectd/collectd.git
#cd collectd
#git checkout fff795c9846bd8fe4bc7f76bcd83a2b8cefb4525
#cp ../collectdpatch/telemetry-add-collectd-plugin-patch.patch ./
#patch -p1 < telemetry-add-collectd-plugin-patch.patch
```

2. Build and install.

```
#./build.sh
#./configure --enable-write_prometheus
#make -j && make install
NOTE: Collectd will be installed by default in /opt/collectd
```

3. Configure Collectd.

   Note: collectd.conf is optimised for 16 vBNG instances because our example deploys 8 vBNG instances on socket 1.

```
#cd ../collectdpatch/
```

   a. Comment out socket 0 in collectd.conf.

   b. Copy the config file over to /opt/collectd/etc.

```
cp collectd.conf /opt/collectd/etc/
```

   c. Confirm that the following lines are uncommented in order to load the following plugins:

```
vim  /opt/collectd/etc/collectd.conf
LoadPlugin syslog
LoadPlugin logfile
LoadPlugin csv
LoadPlugin dpdk_telemetry

# Example of the dpdk_telemetry stanza for both uplink and
downlink for 2 vBNG instances.
# Collectd.conf should contain 8 vBNG instances & will have 16
<Host> blocks configured.
<Plugin dpdk_telemetry>
#########################################################Instance 0
on socket 0
   <Host "ip_pipeline_ul_0_0">
      ClientSocketPath "/var/run/.telemetry_client_ul_0_0"
      DpdkSocketPath "/var/run/dpdk/ip_pipeline_ul_0_0/telemetry"
   </Host>
   <Host "ip_pipeline_dl_0_0">
      ClientSocketPath "/var/run/.telemetry client dl 0 0"
      DpdkSocketPath "/var/run/dpdk/ip_pipeline_dl_0_0/telemetry"
   </Host>
#########################################################Instance 0
socket 1
   <Host "ip_pipeline_ul_1_0">
      ClientSocketPath "/var/run/.telemetry_client_ul_1_0"
      DpdkSocketPath "/var/run/dpdk/ip_pipeline_ul_1_0/telemetry"
```

```
   </Host>
   <Host "ip_pipeline_dl_1_0">
      ClientSocketPath "/var/run/.telemetry_client_dl_1_0"
      DpdkSocketPath "/var/run/dpdk/ip_pipeline_dl_1_0/telemetry"
   </Host>
#####################################################################
```

d. Copy the Application types database.

```
cp <Your application>_types.db /opt/collectd/share/collectd/
```

e. Ensure the following lines are present in Collectd's config file (under line 18):

```
vim  /opt/collectd/etc/collectd.conf
TypesDB     "/opt/collectd/share/collectd/types.db"
TypesDB     "/opt/collectd/share/collectd/vbng_types.db"
```

f. The Collectd interval is set to 10s by default. Set it according your configuration's requirements (line 43).

```
Interval    1
```

4. Run Collectd.

```
# NOTE: Before  deploying Collectd, ensure traffic is
successfully returning to your Traffic Generator.
# If, for any reason, one of the vBNG Pods crash, all the vBNG
Pods will need to be restarted, as well as the Collectd daemon
as, due to a Telemetry API and plugin limitation,
# Collectd is unable to reconnect to the Pod which crashed and
restarted.

cd /opt/collectd
sbin/collectd -f
```

5. Confirm that the stats are being reported and written to CSV files.

```
                        ls -lisa
/opt/collectd/var/lib/collectd/vbng_X_ul/dpdk_telemetry-upstream/
                        ls -lisa
/opt/collectd/var/lib/collectd/vbng_X_dl/dpdk_telemetry-
downstream/
```

6. Deploy a cron job to clean up old telemetry stats.

```
crontab -e
Add following line to file : 0 0 * * * rm -rf
/opt/collectd/var/lib/collectd/*
crontab -l to ensure changed has been added
```

7. Enable and configure Prometheus endpoint by adding to collectd.conf.

```
LoadPlugin write_prometheus
<Plugin write_prometheus>
        Port "9103"
</Plugin>
```

3.7 Traffic generator setup

3.7.1 Prepare traffic generator on server 2

Configure boot settings similar to Server 1 and install required Linux* packages.

1. Install packages as follows using the Linux package manager.

```
# apt-get install build-essential
# apt-get install libpcap-dev
# apt-get install dtach
```

Note: Python packages already installed can be listed using pip list.

```
# pip install fabric -U --force-reinstall
# pip install paramiko -U --force-reinstall
# pip install cryptography -U --force-reinstall
```

Note: Install the following Intel® Network Adapter Drivers for PCIe.

```
# i40e-2.7.12
# wget https://downloadmirror.intel.com/28381/eng/i40e-
2.7.12.tar.gz
# tar -zvf i40e-2.7.12.tar.gz
# cd i40e-2.7.12/src/
# make install
# loading the new module:
  rmmod i40e; modprobe i40e

# i40evf-3.6.10
# wget https://downloadmirror.intel.com/28382/eng/i40evf-
3.6.10.tar.gz
# tar -zvf i40evf-3.6.10.tar.gz
# cd i40e-3.6.10/src/
# make install
# loading the new module:
  rmmod i40evf; modprobe i40evf
```

3.7.2 Traffic gen server environment setup

Note: This example uses a virtual Broadband Network Gateway (vBNG) reference application to showcase the resiliency of the workload with closed loop failover capability, however the solution itself is application-agnostic.

1. Get traffic gen source for your application and untar the source.
2. Configure the vBNG Pkt-gen environment. Refer to Section Traffic Generator Configuration Script4.0 for the full pktgen-config.sh. script.
   a. Set the PKTGEN_HOST environment variable.

```
# export PKTGEN_HOST=y
```

   b. Go to vBNG directory and update pktgen-config.sh for the local environment.

```
# cd /root/vBNG
```

```
# vi pktgen-config.sh
```

c. Run the main environment script which also runs pkgen-config.sh.

```
# source $MYHOME/env.sh
```

3. Download and build DPDK v18.11.

```
# cd $MYHOME
# git clone http://dpdk.org/git/dpdk
# cd dpdk
# git checkout v18.11
```

4. Ensure the DPDK environment variable settings are correct in $MYHOME/pktgen-config.sh.

```
export RTE_SDK=$MYHOME/dpdk
export DPDK_DIR=$RTE_SDK
```

5. Build DPDK.

```
# cd $DPDK_ROOT_RELEASE
# make install T=x86_64-native-linuxapp-gcc
# export RTE_TARGET=x86_64-native-linuxapp-gcc
# export DPDK_BUILD=$RTE_SDK/$RTE_TARGET
```

6. Download and Build DPDK Pkt-gen and download and checkout PktGen v3.5.4.

```
# cd $MYHOME
# git clone http://dpdk.org/git/apps/pktgen-dpdk
# cd pktgen-dpdk
# git checkout pktgen-3.5.4
```

7. Patch Pktgen.

```
# cd $PKTGEN_ROOT
# patch -p1 < $MYHOME/dpdk_18.11_patches/v1-0001-pktgen-mac-
set.patch
```

Note: Ignore the whitespace warnings that are reported.

8. Build Pkt-gen.

```
# make
```

9. Build PF Init Application.

Note: Ensure you have run source env.sh.

```
# source $MYHOME/env.sh
# build_pf_init_app
```

10. Bind PFs to DPDK.

a. Source env.sh.

Note: Ensure you have run source env.sh.

```
# source $MYHOME/env.sh
```

b. Load igb_uio kernel module.

```
# ins
```

c. Bind PFs.

```
# bind_pf_dpdk
```

11. Create VFs.

```
# create_vfs_dpdk
```

12. Bind VF Ports to DPDK on vBNG Pkt-gen server.

Note: NIC PF port addresses have already been set in pktgen-config.sh.

Bind NIC VFs:

```
# bind_vf_dpdk
```

Note: Function check_ports_bound_status can be used to check if ports needed have bound to DPDK successfully.

```
# check_ports_bound_status
```

Note: Modify your pktgen application in such way that it can start the traffic in PF's for both sockets.

13. Run Pkt-gen PF init app.

```
# running pf init for pktgen takes in two arguments
  #1: is the position of the pf in pf array in pktgen-config.sh
  #2: is the base instance to start at (used in mac addressing)
# Example to run Pktgen Instance 0 and then run Pktgen Instance 1
```

Note: Running per PF brings up and sets MAC addresses for both DL and UL VFs for the corresponding PF.

```
# run_pf_init_0 0 0         ## this is to initialize the pf on
socket 0
# run_pf_init_1 1 0         ## this is to initialize the pf on
socket 1
```

Note: To run in background:

```
# run_pf_init_0 0 0 &
# run_pf_init_1 1 0 &
```

14. Run 4 instances of pktgens as below. 2 are for instance A (Instance in socket 0) and 2 are for instance B (Instance in socket 1).

Instance A:

```
run_pktgen_with_pcap_0 0 vbng_dl_r_4k_pppoe.pcap d
run_pktgen_with_pcap_0 0 vbng_ul_r_4k_pppoe.pcap u
```

Instance B:

```
run_pktgen_with_pcap_1 0 vbng_dl_r_4k_pppoe.pcap d
run_pktgen_with_pcap_1 0 vbng_ul_r_4k_pppoe.pcap u
```

15. Run pktgen_simpl.py to start/stop the traffic on pktgen sessions.

```
# Start traffic on instance A
./vbng_traffic_cmd.py 0 1
# Stop traffic on instance A
./vbng_traffic_cmd.py 0 0
# Start traffic on instance B
./vbng_traffic_cmd.py 1 1
# Stop traffic on instance B
./vbng_traffic_cmd.py 1 0
```

3.8 Alertmanager

Alertmanager will take care of triggering webhooks or distribute notification via mail when alert occurs. Deploy it on Server 1.

Download the alertmanager package (version 0.16.1 or later), unpack it, and go into package folder:

```
cd $WORKSPACE
wget
https://github.com/prometheus/alertmanager/releases/download/v0.16.1/alertmanager-0.16.1.linux-amd64.tar.gz
tar xzf alertmanager-0.16.1.linux-amd64.tar.gz
cd alertmanager-0.16.1.linux-amd64
```

Configure alertmanager by modifying webhook to point to alerthandler host on port 29000.

You can also modify groups timings according to your needs. For more information, refer to: https://prometheus.io/docs/alerting/configuration/

Example `alertmanager.yml`:

```
global:
  resolve_timeout: 5m

route:
  group_by: ['alertname']
  group_wait: 1s
  group_interval: 30s
  repeat_interval: 1m
  receiver: 'web.hook'
receivers:
- name: 'web.hook'
  webhook_configs:
  - url: 'http://<alerthandler_host>:29000/'
inhibit_rules:
  - source_match:
      severity: 'critical'
    target_match:
      severity: 'warning'
    equal: ['alertname', 'dev', 'instance']
```

Run alertmanager with:

```
./alertmanager --config.file="alertmanager.yml"
```

3.9 Prometheus*

Prometheus scrapes telemetry from collectd, stores it and evaluates against rules to create an alert when an issue occurs. Deploy it on Server 1.

Download the Prometheus package (version 2.7.2 or later), unpack it, and go into package folder:

```
cd $WORKSPACE
wget
https://github.com/prometheus/prometheus/releases/download/v2.7.2
/prometheus-2.7.2.linux-amd64.tar.gz
tar xzf prometheus-2.7.2.linux-amd64.tar.gz
cd prometheus-2.7.2.linux-amd64
```

Configure Prometheus by adding alertmanager url, including rules file and pointing the scrape job to collectd. To react faster, change the scrape and evaluation interval to 1s.

For more information, refer to:
https://prometheus.io/docs/prometheus/latest/configuration/configuration/

Example `prometheus.yml`:

```
# Global config
global:
  scrape_interval:     1s # Default is every 1 minute.
  evaluation_interval: 1s # Evaluate rules. The default is every
1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      - 127.0.0.1:9093  # alertmanager host

# Load rules once and periodically evaluate them according to the
global 'evaluation_interval'.
rule_files:
  - "example_rules.yml"

# A scrape configuration
scrape_configs:
  - job_name: 'collectd'
    static_configs:
    - targets: ['localhost:9103']
```

Define an alert rule. In our scenario, we watch for the number of corrected memory errors to reach 5 in the previous 5 seconds. For more information, refer to:
https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

Example `example_rules.yml`:

```
groups:
- name: alert.rules
  rules:
  - alert: mcelog_memory_error
```

```
    expr:
increase(collectd_mcelog_errors_total{type="corrected_memory_erro
rs", mcelog="SOCKET_0_CHANNEL_0_DIMM_any"}[5s]) > 5
    labels:
      severity: "critical"
    annotations:
      summary: "corrected memory error occured"
      description: "corrected memory error occured on {{
$labels.mcelog }} at {{ $labels.exported_instance }}"
```

Run Prometheus with:

```
./prometheus --config.file="prometheus.yml"
```

3.10 AlertHandler

AlertHandler implements automated responses to telemetry-based alerts enabling the system to adapt to state change. It listens on a port, waiting for incoming JSON packet describing alerts via webhook. On receiving an alert, it triggers an action or a user-configured script. Deploy it on Server 2.

Clone the alertHandler repository and build it:
https://github.com/intel/alert-handler-for-custom-metrics

```
cd $WORKSPACE
git clone https://github.com/intel/alert-handler-for-custom-
metrics
cd alert-handler-for-custom-metrics
go build
```

Configure alertHandler to listen on port (29000) for particular alerts (mcelog_memory_error) and bind them with triggering proper scripts (action.sh).

Example `alert-handler-config.json`:

```
{
    "port":":29000",
    "url-path":"/",
    "script-directory" :"scripts/",
    "alerts" : {
            "mcelog_memory_error" : {
                "name" : "mcelog_memory_error",
                "summary":"mcelog_memory_error",
                "status": "firing",
                "script-type": "bash",
                "script-name" : "action.sh",
                "args": ["arg1","arg2"]
            }
        }
}
```

Run alertHandler with:

```
./alertHandler
```

3.11 Trigger error scenario

To trigger the error scenario, we inject memory errors into socket 0 of the system using a script. The number of errors to be injected can be passed as a parameter while running the script. The Prometheus alert manager is notified if the number of errors goes beyond the threshold (≥5) in last 5 seconds.

```
root@<Hostname>:~# cat trigger_mem_err_s0.sh
#!/bin/bash
# Copyright (C) 2019 Intel Corporation
# SPDX-License-Identifier: MIT
# checks for input parameter
if [[ $# -ne 1 ]]; then
    echo "Invalid number of parameters (expected number of errors
to inject)"
    exit 1
fi
num_of_errors=$1
if ! [[ ${num_of_errors} =~ ^[0-9]+$ ]]; then
    echo "Invalid parameter (expected number of errors to
inject)"
    exit 1
fi

echo "Injecting $num_of_errors errors ..."

cd /root/mce-inject/  # configure place of your mce-inject
modprobe mce-inject
for (( c=1; c<=$num_of_errors; c++ ))
do
    echo "Injecting error $c"
    ./mce-inject test/corrected_s0
done
```

How to run the script:

```
#./trigger_mem_err_s0.sh <number of errors to be injected>
```

3.12 Remediation action

Remediation action is triggered when a threshold is breached and the system should move the service traffic from instance A on socket 0 to instance B on socket 1. In this setup, when the alert handler receives an alert from the alert manager in Server 1, it triggers a remediation action script (action.sh) that stops the traffic towards instance A and starts traffic in instance B, simulating reroute of service traffic.

```
root@ag13-15-clx:~# cat /root/alertHandler/scripts/action.sh
#!/bin/bash
# Copyright (C) 2019 Intel Corporation
# SPDX-License-Identifier: MITexec 3>&1 1>>/root/alertHandler/log
2>&1
cd /root/
```

```
date
python vbng_traffic_cmd.py 0 0  ## to Stop traffic on instance A
(Socket 0)
echo ''
date
python vbng_traffic_cmd.py 1 1  ## to Start traffic on instance B
(Socket 1)

echo ''
root@ag13-15-clx:~#
```

The remediation action script internally calls a pktgen script that looks for the pktgen socket number opened and performs a traffic start/stop.

```
# cat vbng_traffic_cmd.py

"""
vbng_traffic_cmd.py <instance_num> <start/stop>

This script is example of how you can manage traffic of
your pktgen instances running in warm standby mode.
"""

import socket
import sys

# check input args
if len(sys.argv) != 3:
    print('invalid number arguments, expected instance_num[0-1] and action[0-stop,
          1-start]')
    sys.exit(1)
for i in range(1, 3):
    if sys.argv[i] not in ['0', '1']:
        print('invalid arguments, expected instance_num[0-1] and action[0-stop, 1-
           start]')
        sys.exit(1)

INSTANCE = int(sys.argv[1])  # 0/1
ACTION = int(sys.argv[2])  # 0 = stop / 1 = start

START_RATE = "pktgen.start(\"all\");"
STOP_RATE = "pktgen.stop(\"all\");"
PKTGEN_HOSTNAME = "127.0.0.1"

MY_PORTS = [[8086, 8094], [8081, 8090]]  # configure your pktgen instances ports
          here
SOCKET_ARRAY = []

print("Executing Rate change to pktgens on:" + PKTGEN_HOSTNAME)

# open the socket ports for the UL and DL pktgen
for i in range(2):
    print("Opening Socket", PKTGEN_HOSTNAME, MY_PORTS[INSTANCE][i])
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    sock.connect((PKTGEN_HOSTNAME, MY_PORTS[INSTANCE][i]))
    SOCKET_ARRAY.append(sock)
print(SOCKET_ARRAY)

# send in the start/stop command
for i in range(2):
    if ACTION == 0:
        SOCKET_ARRAY[i].sendall(STOP_RATE)
    else:
        SOCKET_ARRAY[i].sendall(START_RATE)

# Close the ports
for i in range(2):
    SOCKET_ARRAY[i].shutdown(1)
    SOCKET_ARRAY[i].close()
```

4.0 Traffic Generator Configuration Script

This script is provided for reference only.

```
# cat pktgen-config.sh
#!/usr/bin/env bash

###########################################################################
####                      PKTGEN HOST CONFIGURATION                    ####
###########################################################################

###########################################################################
####                    ENVIRONMENT VARIABLED FOR PKTGEN SETUP         ####
###########################################################################

# Root directory where files have been unpacked
export MYHOME="/root/vBNG"

export DPDK_HOME="/root/vBNG/dpdk"

export RTE_SDK=$DPDK_HOME
export RTE_TARGET=x86_64-native-linuxapp-gcc
export DPDK_DIR=$RTE_SDK
export DPDK_BUILD=$RTE_SDK/$RTE_TARGET
export DPDK_ROOT_RELEASE="$DPDK_HOME"
export DPDK_ROOT_DEBUG="$DPDK_HOME/dpdk-dbg"

export PKTGEN_ROOT="$MYHOME/pktgen-dpdk"

export VBNG_ROOT="$MYHOME/vbngd"

# This directory contains pcaps
export PCAP_ROOT_DL="$MYHOME/pcap/dl_pcaps"

export PCAP_ROOT_UL="$MYHOME/pcap/ul_pcaps"

# Pktgen cores to use on socket 0, first is master core
export CORES_s0="1-27,57-83"

# Pktgen cores to use on socket 1, first is master core
export CORES_s1="29-55,85-111"

# Set to include Cores
#export PKTGEN_CORE_LIST="$CORES_s0" # uncomment if using just socket 0
#export PKTGEN_CORE_LIST="$CORES_s1" # uncomment if using just socket 1
export PKTGEN_CORE_LIST_0="$CORES_s0" # uncomment if using both sockets
export PKTGEN_CORE_LIST_1="$CORES_s1" # uncomment if using both sockets

# Memory on both sockets
export PKTGEN_SOCKET_MEM="1024,1024"
```

```
### Memory to be used for PCAP pkt size scaling test ###
#export PKTGEN_SOCKET_MEM="2048,0"

#export PKTGEN_SOCKET_MEM="1024,0"

### Set Application Type ###
# set to 1 for BNG or set to 2 for pkt_fwd
export APP_TYPE=1

### Set the base instance traffic generator is starting at ###
export BASE_INSTANCE=0

#############################################################################
####                    PKTEGN INSTANCE ASSIGNMENT                       ####
####                    SETUP FOR 8 DL & 8 UL Instances                  ####
#############################################################################

# PF ports to use on socket 0
declare -a pf_ports_s0=("18:00.0" "18:00.1")

# PF ports to use on socket 1
declare -a pf_ports_s1=("af:00.0" "af:00.1")

# VF Ports for DL to use if using socket 0
declare -a vf_ports_dl_s0=("18:02.0" "18:0a.0")

# VF Ports for UL to use if using socket 0
declare -a vf_ports_ul_s0=("18:02.1" "18:0a.1")

# VF Ports for DL to use if using socket 1
declare -a vf_ports_dl_s1=("af:02.0" "af:0a.0")

# VF Ports for UL to use if using socket 1
declare -a vf_ports_ul_s1=("af:02.1" "af:0a.1")

# Socket 0 Ports to use for Downlink
declare -a pktgen_ports_dl_s0=("8086" "8087" "8088" "8089")

# Socket 0 Ports to used for Uplink
declare -a pktgen_ports_ul_s0=("8094" "8095" "8096" "8097")

# Socket 1 Ports to use for Downlink
declare -a pktgen_ports_dl_s1=("8081" "8082" "8083" "8084")

# Socket 1 Ports to used for Uplink
declare -a pktgen_ports_ul_s1=("8090" "8091" "8092" "8093")

# Downlink Pktgen instances running on socket 0 cores
declare -a pktgen_inst_cores_dl_s0=("[2:58].0" "[3:59].0")

# Uplink Pktgen instances running on socket 0 cores
declare -a pktgen_inst_cores_ul_s0=("[4:60].0" "[5:61].0")
```

```
# Downlink Pktgen instances running on socket 1 cores
declare -a pktgen_inst_cores_dl_s1=("[30:86].0" "[31:87].0")

# Uplink Pktgen instances running on socket 1 cores
declare -a pktgen_inst_cores_ul_s1=("[32:88].0" "[33:89].0")

### Set num of VFS for pktgen
export NB_VFS=2

##############################################################################
##############################################################################
##############################################################################

##############################################################################
####    COMBINE ALL ARRAYS INTO 1 COMMON ARRAY FOR SETUP                  ####
####    DECLARE PCAP FILE DIRECTORIES                                     ####
##############################################################################

# PF Ports for both sockets
declare -a pf_ports=()
pf_ports+=("${pf_ports_s0[@]}" "${pf_ports_s1[@]}" )

# Downlink VF Ports for both sockets
declare -a vf_ports_dl=()
vf_ports_dl+=( "${vf_ports_dl_s0[@]}" "${vf_ports_dl_s1[@]}" )

# Uplink VF Ports for both sockets
declare -a vf_ports_ul=()
vf_ports_ul+=( "${vf_ports_ul_s0[@]}" "${vf_ports_ul_s1[@]}" )

# Downlink Cores for both sockets
declare -a pktgen_inst_cores_dl=()
pktgen_inst_cores_dl+=("${pktgen_inst_cores_dl_s0[@]}"
          "${pktgen_inst_cores_dl_s1[@]}" )

# Uplink Cores for both sockets
declare -a pktgen_inst_cores_ul=()
pktgen_inst_cores_ul+=("${pktgen_inst_cores_ul_s0[@]}"
          "${pktgen_inst_cores_ul_s1[@]}" )

# PCAPS: a list of pcap files that can be used for both DL & UL
declare -a pcaps_dl=(`cd ${PCAP_ROOT_DL} && ls -1 *.pcap`)

declare -a pcaps_ul=(`cd ${PCAP_ROOT_UL} && ls -1 *.pcap`)
# echo $pcaps

##############################################################################
##############################################################################
##############################################################################

##############################################################################
####                    PF INIT APP SETTINGS                              ####
```

```
################################################################################
export PF_INIT_LCORE_0=27
export PF_INIT_COREMASK_0=0x8000000
export PF_INIT_SOCKET_MEM_0="1024,0"
export PF_INIT_SOCKET_MEM_LIMIT_0="1024,1"
export PF_INIT_LCORE_1=55
export PF_INIT_COREMASK_1=0x80000000000000
export PF_INIT_SOCKET_MEM_1="0,1024"
export PF_INIT_SOCKET_MEM_LIMIT_1="1,1024"
export PF_INIT_NUM_DL_VFS=1
export PF_INIT_NUM_UL_VFS=1
################################################################################
################################################################################
################################################################################
```

5.0 Summary

This document has described a closed loop resiliency demo that uses a virtual Broadband Network Gateway (vBNG) reference application to showcase closed loop failover capability. The solution itself is application-agnostic and other workloads can be implemented.

It is now becoming a strategic imperative for Comms Service Providers to automate their networks. With the exponential growth in devices that must be managed in the network, closed loop automation is required for efficiency gains, cost reduction, and most importantly productivity improvement. For CSPs, ensuring the customer experience remains impeccable is a top priority. They need to be able to minimise network downtime and increase service availability.

With this demo, closed loop failover capability is now possible. The demo uses Intel server platform features and metrics to identify when issues occur and to immediately react to limit any outage time. Closed loop systems are the starting point for tomorrow's next-generation self-healing and self-optimizing networks.